Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

**Отчёт по индивидуальной работе №1**

Дисциплина: Транслирующие системы

Тема: Программирование лексического анализатора на языке lex

Выполнил студент гр. 5130901/20103 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А. Вагнер

(подпись)

Принял старший преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.Н. Цыган

(подпись)

“ ” 2024 г.

Санкт-Петербург

2024

Оглавление

[Текст задания 3](#_Toc180688496)

[Ход работы 3](#_Toc180688497)

[Вывод 10](#_Toc180688498)

Спи

[Рис. 1 Константы 3](#_Toc180688830)

[Рис. 2 Регулярные выражения и связанные скрипты 4](#_Toc180688831)

[Рис. 3 – работа программы с -d 5](#_Toc180688832)

[Рис. 4 – Функции обработки OP и REAL 6](#_Toc180688833)

[Рис. 5 – Обнаружение ошибок 6](#_Toc180688834)

[Рис. 6 – Тестирование программы 7](#_Toc180688835)

[Рис. 7 – Фрагмент из попыток воспроизвести ошибку 9](#_Toc180688836)

[Листинг 1 – Содержание файла ind.l 10](#_Toc180688837)

Текст задания

Спроектировать LEX-программу, которая в контексте операций сравнения языка FORTRAN выясняет, какое чисто задано – целое или вещественное и переводит синтаксически корректные предложения фортрана в эквивалентные предложения языка C.

Ход работы

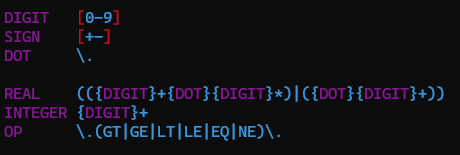


Рис. Константы

Сперва рассмотрим выбранные константы шаблонов. Константа DIGIT соответствует любой цифре, SIGN – знаку плюс либо минус, DOT – точка и OP – любой из операторов сравнения на языке FORTRAN . Далее из этих шаблонов были составлены шаблоны чисел REAL – некоторое количество цифр (больше нуля), точка и некоторое количество цифр (можно ноль), либо точка и некоторое количество цифр (больше нуля).

Из этих констант были составлены следующий регулярные выражения.

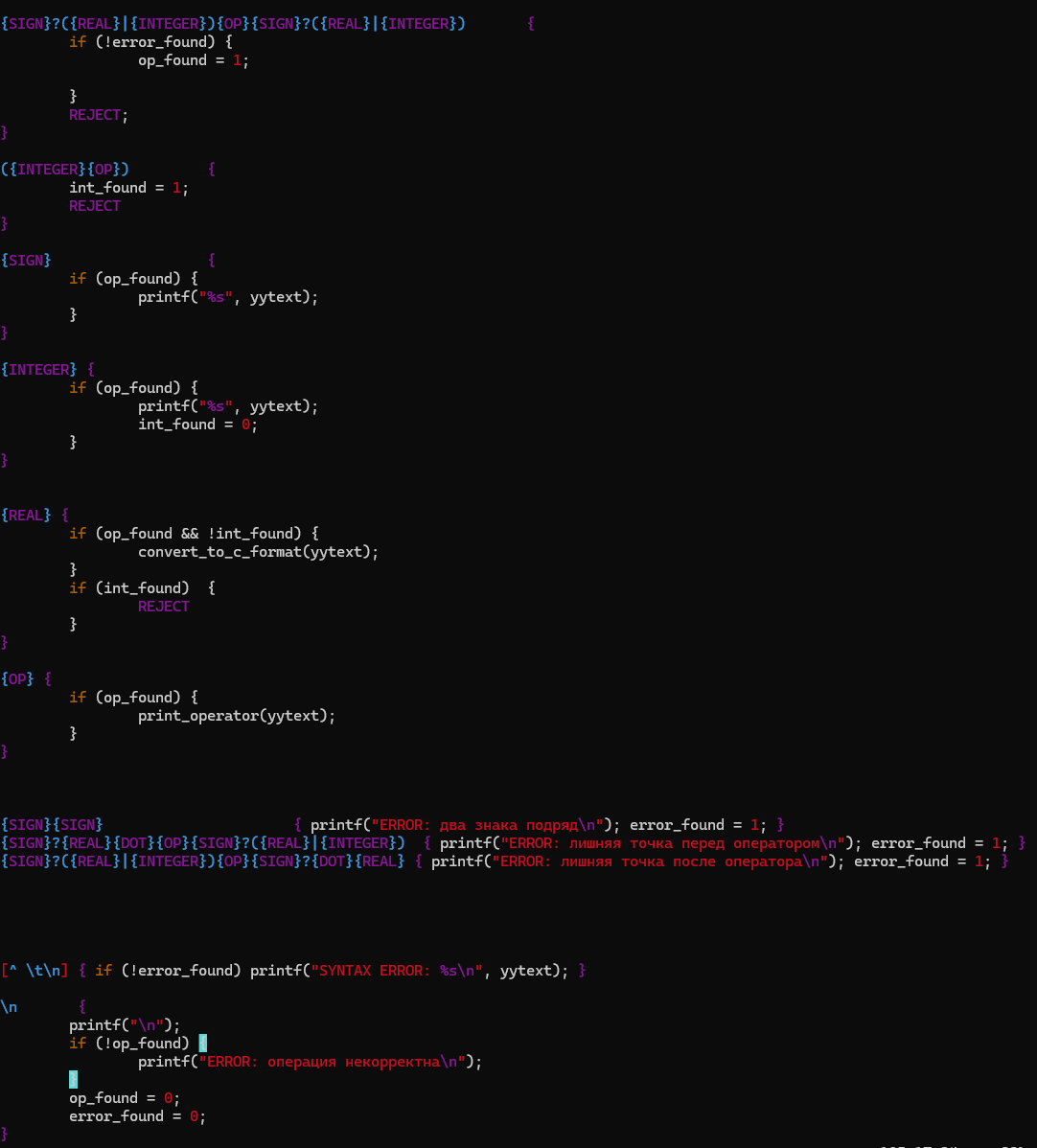


Рис. Регулярные выражения и связанные скрипты

В первую очередь проверяем, соответствует ли полученная строка описанной конструкции из двух чисел со знаком или без и операцией сравнения между ними. Если да – изменяем состояние op\_found на 1, что означает, что строка является валидной операцией сравнения и мы можем выводить в консоль обработанные подконструкции.

Далее опишем ожидаемые порядок обработки корректной строки, для упрощения процесса запустим программу с флагом -d, то есть с включённой функцией дебага.

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рис. – работа программы с -d

После обнаружения корректной конструкции программа отлавливает и печатает знак. Далее чтобы ошибочно не отнести точку оператора к числу, проверим является ли первое число Integer. При обнаружении конструкции {Integer}{OP} в 1 переходит флажок int\_found, а строка сбрасывается Reject. После этого непременно обнаруживается число REAL, так как конструкция из цифр и точки всегда встречается в корректной строке, но так как мы уже определили, что число является INTEGER, а точка относится к OP, о чём нам говорит флажок int\_found, это правило также сбрасывается оператором REJECT. Далее все оставшиеся подконструкции строки корректно отлавливаются и обрабатываются соответствующими правилами INTEGER, OP, SIGN, REAL. При этом текст, соответствующий правилам SIGN и INTEGER выводится в консоль без изменений, а REAL и OP обрабатываются так, чтобы они соответствовали правилам языка C.

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рис. – Функции обработки OP и REAL

Далее детальнее рассмотрим правила, отлавливающие и сообщающие об ошибках

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рис. – Обнаружение ошибок

Данные правила сообщают об ошибках при обнаружении таких ошибок как лишние знаки, либо точки в строке. Также они изменяют флажок error\_found, который не позволяет выводить в консоль другие подстроки, если была обнаружена ошибка.

Также сообщается о синтаксической ошибке, если элементы строки не соответствуют ни одному из представленных правил. При обнаружении переноса строки все флажки обновляются, а если в ходе обработки строки не была найдена полноценная операция сравнения, об этом также сообщается.

Проверим программу на разных наборах входных файлов.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. – Тестирование программы

Заметим, что одна из входных строк была обработана некорректно – 1.111.LT.1.1  
К сожалению, попытки воспроизвести ошибку не увенчались успехом ни в одной из десятков итераций, в том числе внутри того же контекста, что и в примере.

Изображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание

Рис. – Фрагмент из попыток воспроизвести ошибку

Следовательно делаем вывод, что ошибка не является критической.

Вывод

В ходе данной работы была составлена и протестирована программа лексического анализа на языке LEX. Были использованы знания о структуре и синтаксисе программ LEX. Результат соответствует спецификации

Листинг 1 – Содержание файла ind.l

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

**int** yywrap(**void**) { **return** **1**; }

**int** error\_found = **0**;

**int** op\_found = **0**;

**int** int\_found = **0**;

**void** **convert\_to\_c\_format**(**char**\* num) {

**if** (num[**0**] == '.') {

printf("0%s", num);

} **else** **if** (num[strlen(num)-**1**] == '.') {

printf("%s0", num);

} **else** {

printf("%s", num);

}

}

**void** **print\_operator**(**char** \*op) {

**if** (strcmp(op, ".GT.") == **0**) {

printf(" > ");

} **else** **if** (strcmp(op, ".GE.") == **0**) {

printf(" >= ");

} **else** **if** (strcmp(op, ".LT.") == **0**) {

printf(" < ");

} **else** **if** (strcmp(op, ".LE.") == **0**) {

printf(" <= ");

} **else** **if** (strcmp(op, ".EQ.") == **0**) {

printf(" == ");

} **else** **if** (strcmp(op, ".NE.") == **0**) {

printf(" != ");

}

}

%}

DIGIT [**0**-**9**]

SIGN [+-]

DOT \.

REAL (({DIGIT}+{DOT}{DIGIT}\*)|({DOT}{DIGIT}+))

INTEGER {DIGIT}+

OP \.(GT|GE|LT|LE|EQ|NE)\.

%%

{SIGN}?({REAL}|{INTEGER}){OP}{SIGN}?({REAL}|{INTEGER}) {

**if** (!error\_found) {

op\_found = **1**;

}

REJECT;

}

({INTEGER}{OP}) {

int\_found = **1**;

REJECT

}

{SIGN} {

**if** (op\_found) {

printf("%s", yytext);

}

}

{INTEGER} {

**if** (op\_found) {

printf("%s", yytext);

int\_found = **0**;

}

}

{REAL} {

**if** (op\_found && !int\_found) {

convert\_to\_c\_format(yytext);

}

**if** (int\_found) {

REJECT

}

}

{OP} {

**if** (op\_found) {

print\_operator(yytext);

}

}

{SIGN}{SIGN} { printf("ERROR: два знака подряд**\n**"); error\_found = **1**; }

{SIGN}?{REAL}{DOT}{OP}{SIGN}?({REAL}|{INTEGER}) { printf("ERROR: лишняя точка перед оператором**\n**"); error\_found = **1**; }

{SIGN}?({REAL}|{INTEGER}){OP}{SIGN}?{DOT}{REAL} { printf("ERROR: лишняя точка после оператора**\n**"); error\_found = **1**; }

[^ \t\n] {

**if** (!error\_found) {

printf("SYNTAX ERROR: %s**\n**", yytext);

}

error\_found = **1**;

}

\n {

**if** (!op\_found || error\_found) {

printf("ERROR: операция некорректна");

}

op\_found = **0**;

error\_found = **0**;

printf("**\n\n**");

}

%%

**int** main(**void**) {

yylex();

**return** **0**;

}